Kildekode-systemer

Dette er et samskrivingsdokument skrevet som en del av faget Systemutvikling og dokumentasjon 2019. Dokumentet skal utarbeides av studentene i felleskap samtidig (samskrivning), alle har et felles ansvar for innholdet mtp struktur, utseende, skrivefeil, m.m. Hver student har hovedansvar for å skrive utdypende om et bestemt tema eller et bestemt kildekode-system, ta utgangspunkt i tabellen nedenfor. Man må også kvalitetssjekke en del som er skrevet av en annen.
Dokumentet er ment å gi en kort oversikt over kildekode-systemer og dekke pensum innen dette temaet.

Dokumenter er utviklet som et “Open Source” prosjekt, dvs her er det ikke en sjef som styrer innholdet, men basert på at alle involverte tar ansvaret for både enkeltdeler og helheten.

Deadline:
Etter denne dagen vil det bli generert et PDF dokument som da vil inngå som en del av pensum (kun på et overordnet nivå, ingen detaljer, dvs må vite forskjell på distribuerte og sentraliserte systemer og kjenne til noen SCC systemer i hver kategori på et overordnet nivå)

Hver deltager må velge seg minst et kildekode-system de ønsker å skrive om.

Deltagere (fyll inn navnet deres for den delen dere er ansvarlig for):

Part	Responsible	Status	Quality Control (Name)
Tittelside	HPH	100%	
Innledning	HPH	100%	
Sentraliserte	OD	100%	HPH
Distribuerte	OD	100%	HPH
Skybaserte	OD	100%	HPH
Git	Silje	100%	
Mercurial	Remi	100%	
GitHub	Azadeh		
GitLab	Jan-Erik		

BitBucket	Joakim		
Perforce	Kevin		
Source Forge	Jon Erik	100%	
Monotone	Kristian		
CVS	NPH	100%	
BitKeeper	Audun		
QVCS			
SVK			
Aegis			
Veracity			
Azure DevOps	Adrian	100%	

Dette er bare noen eksempler, dere kan velge å skrive om andre systemer.

På eksamen kan dere forvente en eller flere av følgende spørsmål:

- Et spørsmål om Kildekodesystemer generelt (overordnet hensikt, overordnet virkemåte, m.m).
- Kort forklare de overordnede forskjellen(e) mellom sentraliserte eller distribuerte kildekodesystemer.
- Kunne vite forskjellen mellom selve kildekode systemet (repository) og “webtjenster” som tar i bruk et eller flere kildekodesystemer. Et eksempel her er Git vs. GitHub
- Kunne navngi minimum 5 kildekodesystemer/kildekodetjenester og kunne si om de er av typen sentraliserte eller distribuerte
- Et mer detaljert spørsmål om kildekodesystemet DU har beskrevet i dette dokumentet

Innholdsfortegnelse

Innholdsfortegnelse **2**

Innledning **2**

Kildekodesystemer 2

Software Configuration Management 3

1. Forskjellige typer kildekode-systemer **4**

1.1. Sentralisert kildekode-system 4

1.2. Distribuert kildekode-systemer 5

2. Kildekode-systemer **7**

2.1. Git 7

2.2. Mercurial 8

2.3. Monotone 10

2.4. Azure DevOps Server 10

2.5. Concurrent Versioning System (CVS) 10

2.6 Perforce 11

3. Skybaserte kildekode-systemer **11**

3.1. Azure DevOps Services 11

3.2. GitHub 12

3.3. GitLab 12

3.4. BitBucket 13

3.5. Source Forge 14

Innledning

Kildekodesystemer

Kildekodekontrollsystemer (Engelsk: “Source Code Control”, **SCC**) eller versjonkontrollsystemer er programvare som kan holde orden på de forskjellige versjonene av en eller flere datafiler (typisk kodefiler men også dokumenter, m.m.). Når en fil oppdateres eller forandres, slettes ikke den gamle versjonen, men blir lagret i en database (Engelsk: “repository”) som inneholder tidligere versjoner av filene. Gamle versjoner kan hentes frem hvis det er behov for det.

Git er kanskje det mest kjente og mest brukte kildekode-system i dag.

Normalt installerer man databasen (repository) på en lokal server hos bedriften som bruker kildekode-systemet.

En annen variant er å bruke en skybasert løsning som man enten kan bruke gratis eller abonnerer på, dvs betale en månedlig sum per bruker. **GitHub** er den mest kjente og brukte varianten av en slik skybasert løsning.

Her er eksempler på noen aktuelle kildekode-kontroll-systemer som brukes mye i dag:

- Git
- Azure DevOps (Azure DevOps Server og Azure DevOps Services)
- Subversion
- CVS (Concurrent Versions System)

Her er noen eksempler på **skybaserte** løsninger:

- GitHub
- Azure DevOps Services

Kilder:

- Wikipedia: https://no.wikipedia.org/wiki/Versjonskontrollsystem
- Halvorsen, Hans-Petter: https://www.halvorsen.blog/documents/teaching/courses/software_engineering.php

Software Configuration Management

Kildekode-kontroll-systemer (SCC) inngår i det videre begrepet **Software Configuration Management (SCM)** systemer. SCM systemer har bakt inn mer funksjonalitet utover ren kildekodekontroll. SCM kan i tillegg brukes til f.eks bugrapportering, dvs et system hvor man kan legge inn bugs (feil og mangler med systemet) i systemet som er under utvikling og som holder oversikt over disse. Både VSTS og GitHub kan betegnes som SCM systemer.

Kilder:
[1] Sommerville: Software Engineering, Kapitel 25 Configuration Management
[2] Halvorsen, Hans-Petter:
https://www.halvorsen.blog/documents/teaching/courses/software_engineering.php

1. Forskjellige typer kildekode-systemer

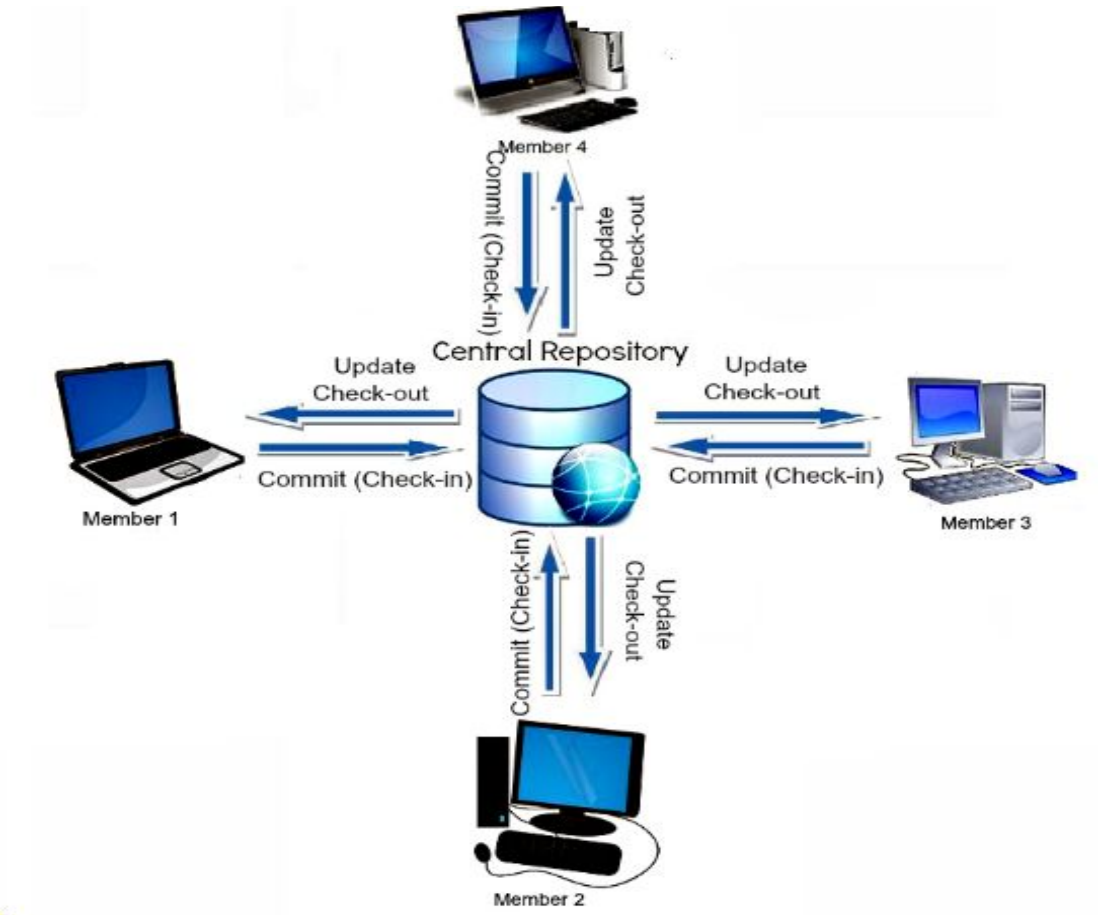
Vi har ulike typer kildekode-systemer, men de kan kategoriserer i to hovedkategorier: sentraliserte- og distribuerte kildekode-systemer. Oppgaven til slike systemer er å lagre versjoner og ulike deler av programmet, slik at utviklerne kan dele disse med hverandre slik at de kan jobbe med ulike deler av programmet samtidig. De kan også brukes til å hente tidligere versjoner.

Kilder:
[1] http://proquest.safaribooksonline.com/book/software-engineering-and-development/deployment/9781787126619/firstchapter#X2ludGVybmFsX0h0bWxWaWV3P3htbGlkPTk3ODE3ODcxMjY2MTkIMkZjaDA0czAyX2h0bWwmcXVlcnk9
[2] https://www.atlassian.com/blog/software-teams/version-control-centralized-dvcs

1.1. Sentralisert kildekode-system

I et sentralisert kildekodesystem jobber programmerere mot et felles diskområde (Central Repository) på en server. Alle programmererne sjekker koden sin ut (Check-out/Update) fra fellesområdet til klient-PC-ene når de skal jobbe videre med koden og inn (Check-in/Commit) til fellesområdet (serveren) når endringer er utført. Utførte endringer blir dermed tilgjengelige for andre. [1].

Programmererne behøver ikke å ha lokale kopier på egne maskiner, da versjonskontrollsystemet holder rede på de ulike kopiene/versjonene. I Figur 1.1 vises en illustrasjon av prinsippet.



Figur 1.1: Illustrasjon av sentralisert kildekodesystem. Figuren er hentet fra scmQuest [1]

Kilder:
[1] scmQuest. https://scmquest.com/centralized-vs-distributed-version-control-systems/. Nedlastet 12.03.2018.

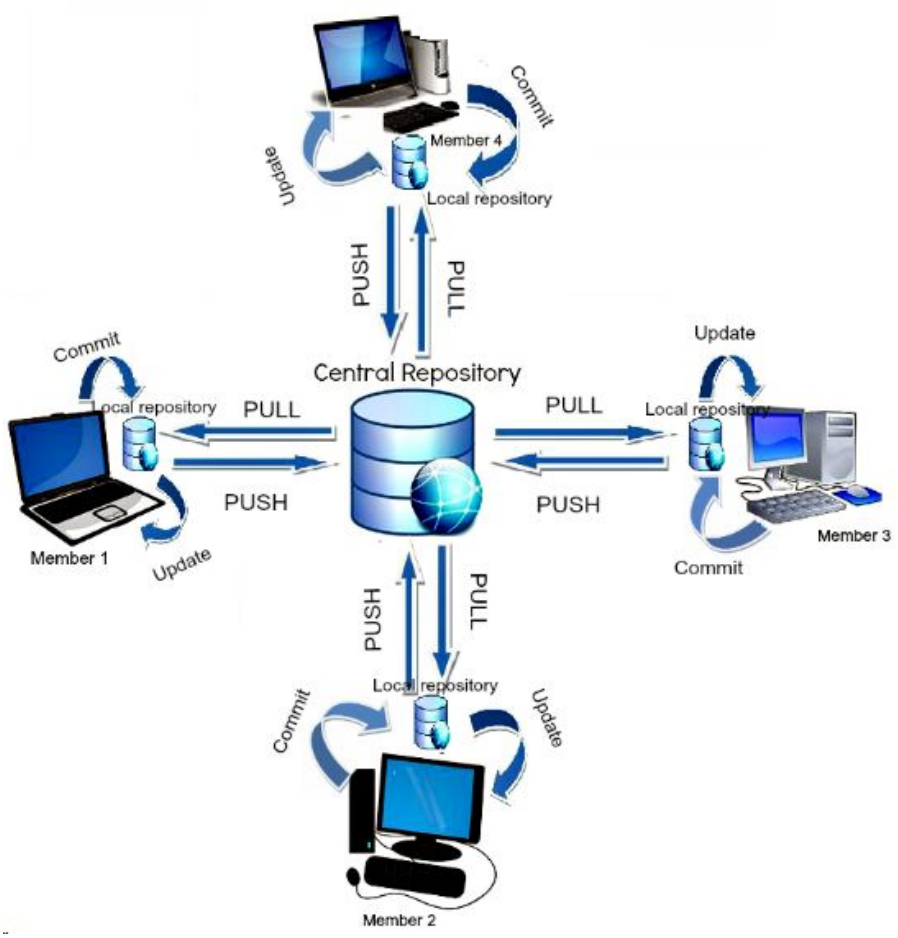
1.2. Distribuert kildekode-systemer

Et distribuert kildekodesystem fordeler lagringen av kildekoden til et prosjekt på flere områder (klienter), i stedet for kun å benytte *ett* felles område (repository). I tillegg benyttes gjerne et sentralt (offisielt) repository.

Ansvar for lagring av versjoner og revisjonshistorikk fordeles dermed på alle som som er del av et felles prosjekt. Den enkelte lagrer altså ikke bare *sin* del av prosjektet, men har på sitt eget, lokale lagringsområde ("repository") en kopi/speiling av hele prosjektet.

Alternativt til en sentralisert løsning, med kun *ett* lagringssted, blir dette dermed en distribuert løsning, en såkalt peer-to-peer løsning, der flere klienter samarbeider om helheten.

Denne løsningen gir også programmererne mulighet for å jobbe offline mot sitt eget "repository", ved å "committe" (lagre varige endringer) til dette". Slike endringer blir først tilgjengelige online for øvrige, ved bruk av en Push-kommando. Tilsvarende henter man ned oppdateringer til klienten online, ved å benytte en Pull-kommando. [1].



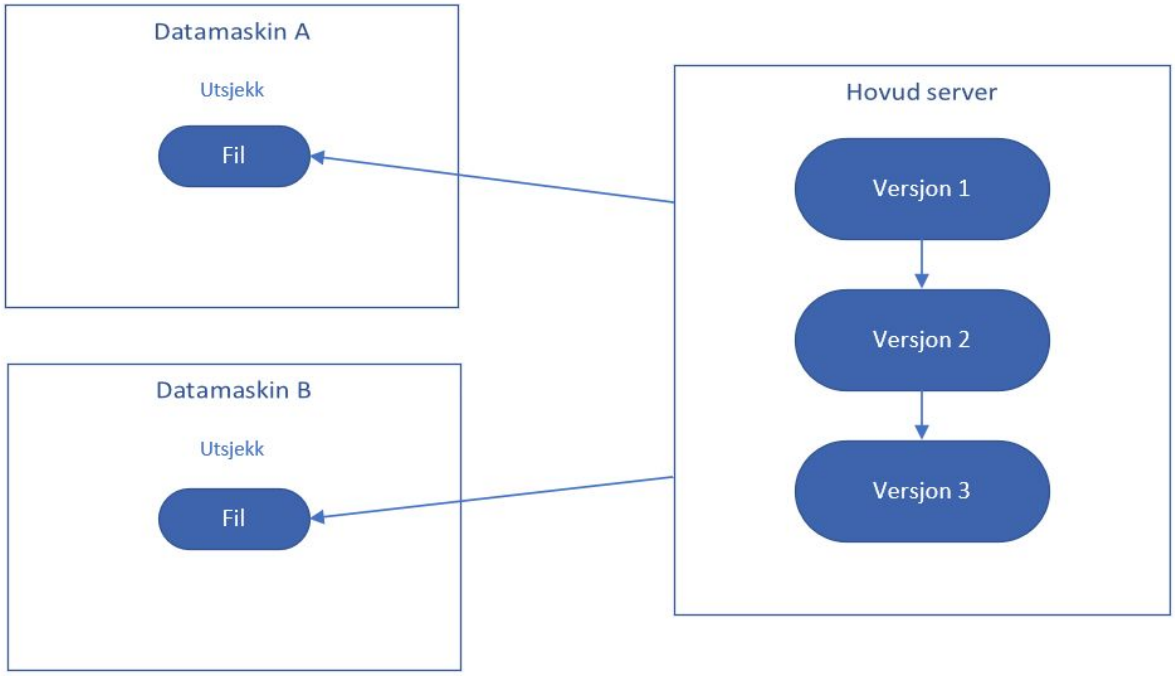
Figur 1.2 Illustrasjon av distribuert kildekodesystem. Figuren er hentet fra scmQuest [1]

Kilder:
[1] scmQuest. https://scmquest.com/centralized-vs-distributed-version-control-systems/. Nedlastet 12.03.2018.

2. Kildekode-systemer

2.1. Git

Git vart utvikla i 2005 av Linus Torvalds, som er grunnleggjaren av Linux. Programmet har seinare vore halde ved like av Junio Hamano. Git er eit verktøy for versjonskontroll og kodedeling. Det vil seie at fleire utviklarar enkelt kan jobbe saman på eit eller fleire prosjekt samtidig. Systemet har versjonskontroll, noko som betyr at heile kodehistorikken blir lagra. Om ein utviklar får ein feil eller eit problem dei ikkje klarer å løyse, kan dei slette sin versjon og hente ned ein ny utan endringane som gjorde at feilen oppstod. Kvar utviklar lasta ned ein kopi av prosjektet, og kan dermed jobbe lokalt på sin eigen pc utan nettilgang. Alle endringane som vert gjort i kopiane vert lasta opp, og teamet kan i fellesskap velje kva endringar dei ønsker å implementera i koden. Figur 2.1 viser korleis to ulike utviklarar kan hente ein versjon av ein fil. Når fila blir sjekka inn, vert det lagra ein ny versjon av ho. Både utviklar A og B versjon 1, når utviklar A lasta opp sin versjon, blir den lagra som versjon 2, og utviklar B si fil blir versjon 3. Teamet kan sjå på alle versjonane og lage ein felles revidert versjon med dei beste endringane frå alle versjonane.



Figur 2.1 Illustrasjon av Git

Kvifor velje Git?
Git har generelt sett høg hastigheit når det kjem til nedlasting og opplasting av filer. Det tek berre nokre sekund om det er mindre fila, dersom fila er større en 1GB vil det ta noko lengre tid. Sidan kvar utviklar kan laste ned sin eigen kopi av prosjektet frå ein felles skybasert oppbevaringssted, gjer det at samarbeid over større geografiske områder lettare. Alle endringar blir lagra, og teamet kan velje den løysinga som er best til den endelege implementeringen. Git er ein gratis teneste, som til dømes kan lastas ned frå https://git-scm.com/download/win. Git blir beskrive som eit enkelt system å komme i gang med, det finst fleire ulike guidar på nett, som til dømes denne: https://git-scm.com/book/en/v2. Git er eit mykje brukt system, noko som gjer at det er relativt lett å finne hjelp og dokumentasjon rundt systemet.

Git og GitHub
Hovudfunksjonen mellom desse to er at Git installerast lokalt på datamaskina, medan GitHub er ein onlinebasert teneste som ein brukar av Git kan bruke til å laste opp eller ned ressursar frå. Meir informasjon om GitHub kan finnast i kapittel 3.2 i dette dokumentet.

Kjelder:
https://www.capgemini.com/no-no/2012/10/hvorfor-bruke-git-i-sma-og-storre-prosjekter/
https://www.ntnu.no/wiki/display/plab/Git+-+versjonskontroll
https://itstud.hiof.no/~tommywm/oppgaver/webutvikling/oblig5/gitgithub.html
https://no.wikipedia.org/wiki/Git
Git nedlasting: https://git-scm.com/download/win
Git guide: https://git-scm.com/book/en/v2
Figur 2.1: laga av SFR

2.2. Mercurial

Mercurial har basis i å være et versjonskontrollert og kommando-linje drevet program som er designet for å være enkelt å bruke, mer spesifikt for tidligere brukere av subversion og CVS vil dette føles som hjemme.

Alle operasjoner i programmet kalles som argumenter til driverprogrammet “hg” (Kjemisk symbol for Kvikksølv). Programmet kommer med et intuitivt og enkelt interface hvor endringer til GUI er tilgjengelig via utvidelser som ønsket. Figur 2-1 under viser enkel Merge funksjon i Mercurial. Her bruker man en “pull” funksjon for å dra en feature inn i hovedkatalogen, dermed “merge” for å legge til endringer og “commit” kommandoen er her brukt i Mercurial som en enkel kommando for verifisering av endring. Ofte gjøres dette ved å først opprette en klone av

prosjektet slik at man ikke mikser ukomplette versjoner. Dette kan gjøres ved bruk av “hg clone project feature1”.

```
hg pull --force <project-to-union-merge>
hg merge
hg commit
```

Figur 2-1: Bildet viser en enkel Merge funksjon.

Mercurial er plattform uavhengig og jobber lokalt på PC hvor hver utvikler kan laste ned kopier av prosjektet og jobbe lokalt. Dette gjør programmet uavhengig av nettverkstilgang eller en lokal server. Det gjør også programmet lokalt, rask og beleilig for brukeren, fordi man enkelt kan lage “diffs” (Kommando for å sammenligne forskjellige revisjoner av filer) mellom revisjoner og enkelt hoppe tilbake i tid for å se på tidligere versjoner av prosjektet. Det er her laget en enkel guide for bruk av Mercurial..

Funksjonaliteten i Mercurial kan oppgraderes og endres ved hjelp av utvidelser, her er det tilgang til en egen wiki-side for dette eller man kan om det ønskes programmere egne utvidelser. Ettersom Mercurial er skrevet hovedsakelig i Python med små deler C av ytelses årsaker, vil alle utvidelser måtte skrives i Python. Funksjonalitet som enkeltbrukere ikke har nødvendigheter for kan man få tilgang til ved hjelp av utvidelser, disse er ikke tilgjengelig for enkeltbrukere grunnet at de regnes som “potensielt program ødeleggende”. Det er også en mulighet for å opprette en to-bro tilkobling mellom Mercurial og Git, via utvidelsen “hg.-git”. Dette gir Mercurial en “Easy to learn and hard to break” struktur.

Kilder:
https://stackoverflow.com/questions/35837/what-is-the-difference-between-mercurial-and-git
https://www.mercurial-scm.org/guide
https://www.mercurial-scm.org/about
https://en.wikipedia.org/wiki/Mercurial

2.3. Monotone

Monotone er et distribuert versjonskontrollsystem. Den gir en enkel filtransaksjonsversjon, med fullstendig offline operasjon og en effektiv samsvarsprotokoll for peer-to-peer. Den forstår historisk følsomme data sammenslåinger, integrert kode anmeldelse og tredjeparts testing. Den bruker kryptering versjon og RSA-sertifikater på klientsiden. Den har god internasjonaliseringstøtte, kjører på Linux, Solaris, Mac OS X, Windows og andre unixes, og er lisensiert under GNU GPL.

Får mere info les .pdf linken i kildene, den er på engelsk.
Nedlastning på Monotone sin hjemmeside i kildene

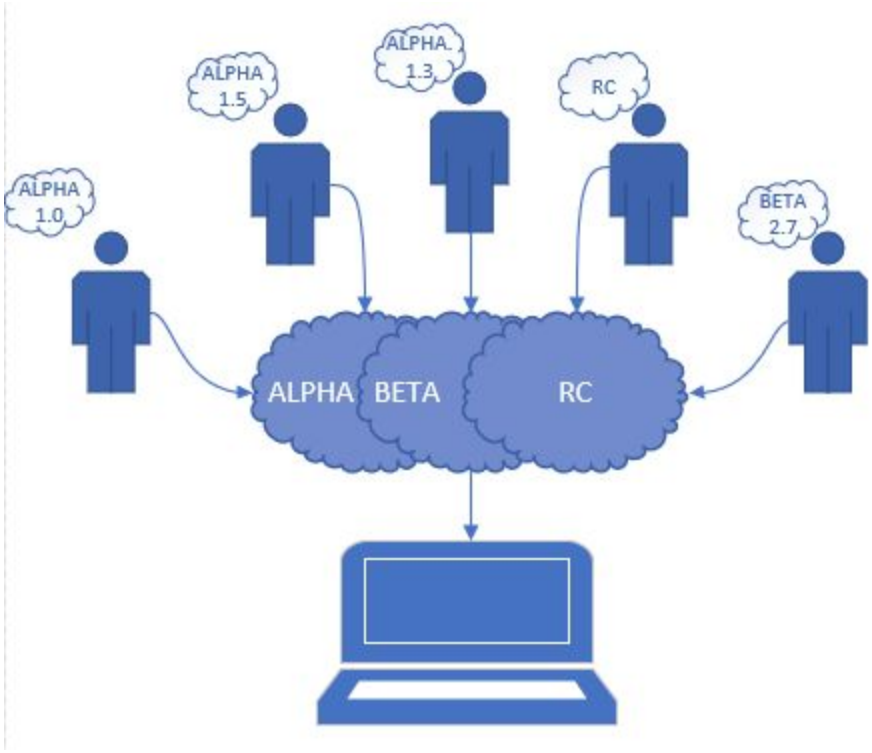
Kilder: https://www.monotone.ca/
https://www.monotone.ca/monotone.pdf

2.4. Azure DevOps Server

Merk forskjellen mellom Azure DevOps Server and Azure DevOps Services.

Kilder:

2.5. Concurrent Versioning System (CVS)



Concurrent Versions System (CVS) er et program som lar en kodeutvikler lagre og hente kildekode fra et prosjekt. Det lar et team av utviklere dele kontrollen over forskjellige versjoner av filer i et felles arkiv. I figuren over er dette beskrev hvor vi har flere utviklere som ønsker å ha tilgang til forskjellige versjoner av prosjektet som er under utvikling. Dette får dem til ved at dem kan gå tilbake til versjonen de ønsker å se. CVS ble opprettet i UNIX og er tilgjengelig i gratis- eller bedrift-versjon. Det er et verktøy som er meget populært på Linux og andre UNIX-baserte systemer. Du kan få tak i CVS på mange måter f.eks. laste ned gratis versjon fra nettet eller fra GNUsine sider, som er et prosjekt for å skape et helt fritt operativsystem: http://ftp.gnu.org/non-gnu/cvs/.

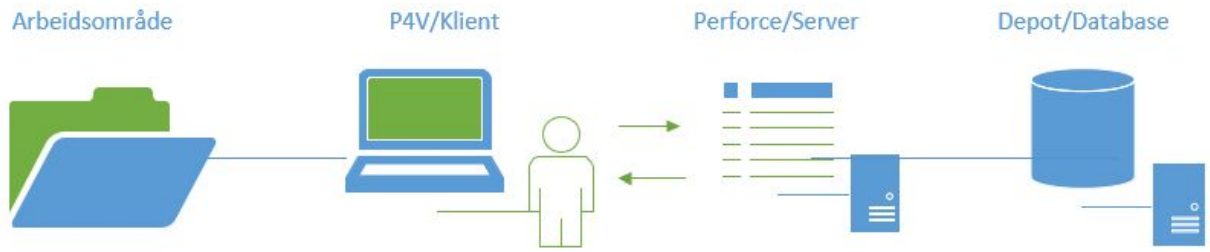
CVS funker på den måten at den ikke lagrer alle versjonen av kildekode filer, men ved å ha en enkelt kopi med oversikt over alle endringene, som er kjent som inkre. Når en utvikler spesifiserer en bestemt versjon, kan CVS rekonstruere den versjonen fra de registrerte endringene. CVS brukes vanligvis til å holde oversikt over hver utviklers arbeid individuelt i en egen arbeidskatalog. Arbeidet til et team av utviklere kan etter ønske slås sammen i en felles deport, et felles område for lagring av data. Endringer fra individuelle lagmedlemmer kan legges til depotet gjennom en «commit»-kommando.
CVS bruker et annet program, Revision Control System (RCS), for å gjøre den faktiske revisjon administrasjonen, det vil si å holde oversikt over endringer som går med hver kildekode fil.

CVS er ikke et byggesystem, et kodekonfigurasjonstyringssystem eller en erstatning for andre gode utviklingspraksis, men heller en måte å styre versjoner av et program som blir utviklet.

Kilder: https://whatis.techtarget.com/definition/Concurrent-Versions-System-CVS (12.03.2019, Kl:14:33)
https://no.wikipedia.org/wiki/GNU (15.03.2019, Kl:12:01)
https://www.gnu.org/software/trans-coord/manual/cvs/cvs.html (15.03.2019, Kl:12:01)

2.6 Perforce

Perforce, eller Perforce Software, er det firmaet som står bak, og eier, Perforce Helix. Perforce Helix er et proprietært kontrollsystem, og versjonkontrollmotor, som ble først skapt i 1995 i Minnesota. Perforce er et verktøy for bedriftversjonadministrasjon og brukes til å håndtere kildefiler og annen dokumentasjon, slik som manualer, nettsider eller administreringsfiler for operativsystem. Alle filer ligger i et "Depot". Filer kan endres i klienten sitt arbeidsområde ("Work Space") og kan laste opp filene i til depoet i form av et "changeset". Depoet har også en oversikt over alle forskjellige versjoner av dokumentene som blir lastet opp. Denne prosessen er illustrert i figur 2.6.1 nedenfor.
P4V som vises i illustrasjonen er Perforce egen applikasjon, men man kan også bruke P4VS som er Microsoft Visual Studio sin versjon.



Figur 2.6.1: Sammehengen mellom klienten og Perforce serverne

3. Skybaserte kildekode-systemer

Skybasert løsninger trenger man ikke installere, da de brukes via nettet. Disse kan enten brukes gratis eller abonneres på, dvs betale en månedlig sum per bruker. **GitHub** er den mest kjente og brukte varianten av en slik skybasert løsning. Andre varianter er Azure DevOps Services som vi bruker i faget Systemutvikling.

Her er noen eksempler på skybaserte løsninger:

- GitHub
- Azure DevOps
- GitLab
- BitBucket

3.1. Azure DevOps Services

Azure Repos er et privat skydriftet Git-repositorium. Git-repositoriumet er et distribuert versjonskontrollsystem. Det støtter enhver Git-klient og det er mulig å sende kode fra IDE, en Git-klient eller et hvilket som helst redigeringsprogram. Man kan gjøre kodesøk enkelt fordi det er kodebevisst som vil si det forstår klasser og variabler [1].

Repos har også TFVC (Team Foundation Version Control) som er et sentralisert versjonskontrollsystem som hjelper brukere med å ha kontroll over endringer man gjør i koden over tid. Mens man endrer kode med TFVC så tas det snapshots som lagres permanent så man kan hente det opp senere om det ønskes. De “historiske” filene vil bare lagres på serveren, og det vil typisk bare lagres en versjon av filen på utviklerens datamaskin [2].

Fordelen med å bruke Azure Repos er at man kan diskutere changesets man gjør med andre personer i samme team på en simpel måte, det er bare å poste en diskusjonstråd og linke til et changeset. Man kan også legge inn flere kommentarer på kode man har skrivd i changesets som reduserer risikoen for å gjøre feil og gjør det mulig å kommunisere med andre hva man har tenkt [1].

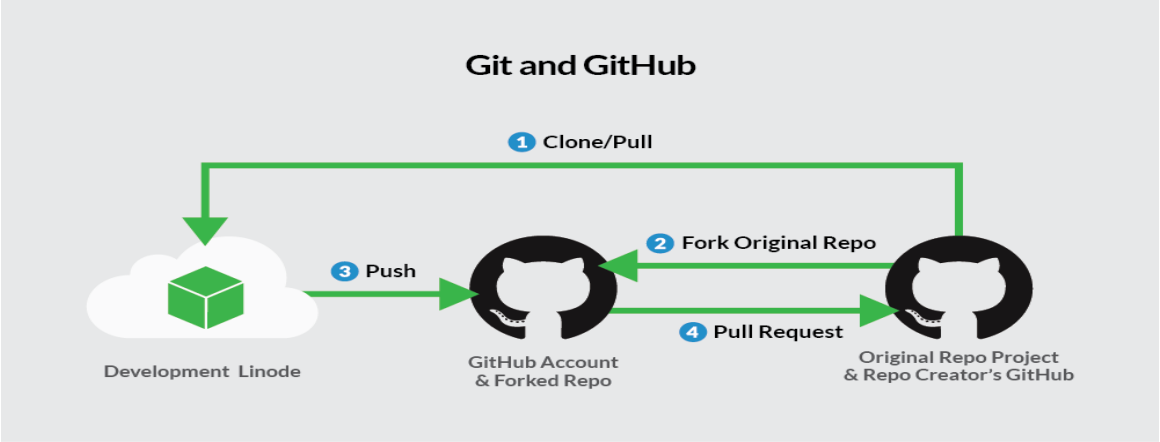
Kilder:
[1]: https://docs.microsoft.com/en-us/azure/devops/repos/tfvc/overview?view=azure-devops
[2]: https://azure.microsoft.com/nb-no/services/devops/repos/

3.2. GitHub

GitHub er en web-basert hosting platform for versjonskontroll ved hjelp av Git, hvor man kan lagre prosjektene sine i nettverk og publiserer de med andre utviklere. Det er mange fordeler ved å bruke GitHub som Repository, Forking og Sosial nettverk osv.

Repository er et lager der hvor man kan lagre alle filene sine for et prosjekt. Derfor alle prosjekter har sine egne repo, man kan få tilgang til det via en unik URL.
Forking skjer når man oppdatere et prosjekt basert på andre prosjekt som allerede hadde befant seg. Hvis man er interessert i å utvikle eller endre noen ting i et prosjekt, må man forklare til repo slik at de endringene man gjør blir ikke lagret som en ny repo.

Man oppdaterer et prosjekt basert på en annen prosjekt og skal utvikle det. Den opprinnelige utviklere kan se alle endringene man gjør og han kan velge om det skal akseptere i det offisielle prosjektet.



Figur 3.2.1- Git og GitHub

GitHubs sosialt nettverk er en veldig viktig egenskap som kan hjelpe oss til å spare tid. På GitHub har hver bruker sine egne profil som viser alle de arbeidene de har gjort gjennom forskjellige prosjekter. Man har mulighet til å diskutere eller samarbeide med andre interesserte utviklere.
I figuren 3.2.1 viser forholdet mellom egenskapene til Github og Git.
Andre fordelen med GitHub er at man kan holde orden på hvem gjør hva, når flere utviklere jobbe sammen i et felles prosjekt.
Kilder:
https://github.com/Oisov/wiki-LKK/wiki/Hvordan-bruke-GitHub
https://en.wikipedia.org/wiki/GitHub
https://www.linode.com/docs/development/version-control/how-to-install-git-and-clone-a-github-repository/

3.3. GitLab

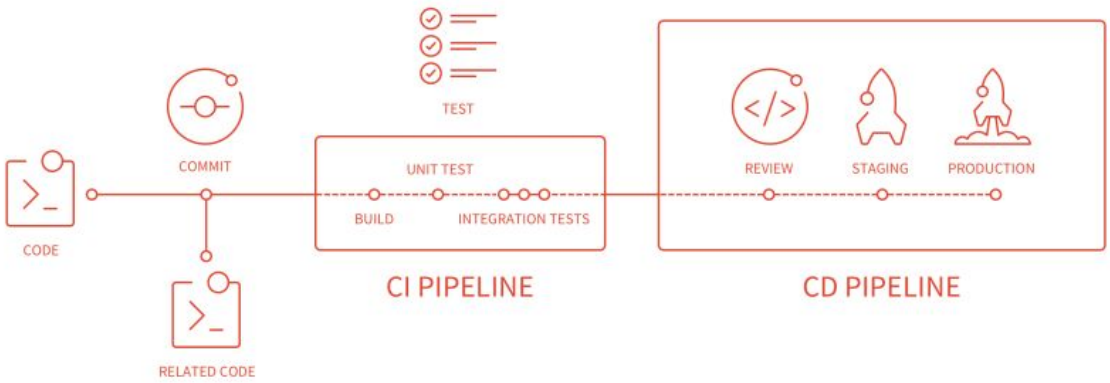
GitLab[1] er et webbasert system for samarbeid i programvareutvikling. GitLab er en "gratis" tjeneste og brukes av store selskaper som f.eks. IBM, Sony, NASA og SpaceX. GitLab tilbyr flere abonnementstyper til ulike priser fra 0 dollar i måneden til 99 dollar i måneden og egenskapene til abonnementet vil variere med prisen[2].

[1] https://en.wikipedia.org/wiki/GitLab
[2] https://about.gitlab.com/pricing/

- Free: $0 månedlig
- Bronze: $4 månedlig
- Silver: $19 månedlg
- Gold: $99 månedlig

Continuous Integration(**CI**)[3] brukes til å samle sammen koden til alle utviklerne i et team og legger det inn i et delt oppbevaringssted. Før koden blir slått sammen og lagt inn i det delte oppbevaringsstedet, blir den kvalitetstestet og validert slik at det blir lette å oppdage feil tidlig i utviklingssyklusen

Continuous Delivery(**CD**) levere den godkjente koden til applikasjonen. Ved å bruke CI og CD kan utviklerne raskt endre/oppdatere applikasjonen. CI blir brukt til oppdage feil og eller redusere antall feil tidlig i utviklingssyklusen og CD leverer endringene (se Figur 3.3.1).



Figur 3.3.1:[4]

3.4. BitBucket

Bitbucket er en web-basert versjonskontroll service. Nettsiden gir brukeren muligheten til å velge mellom git og mercurial for repository og version control. Servicen er gratis, men det er mulig å betale for å låse opp andre funksjoner på nettsiden, slik som IP-whitelisting og two-step verification. Bitbucket er skrevet i Python, og bruker OpenID for innlogging. Nettsiden ble lansert i 2008, eies as Atlassian og er laget av Jesper Nøhr.
Bitbucket kan brukes som skylagring for kodeprosjekter, men har andre funksjoner tilgjengelig, slik som pipelines til issue tracking. Prosjektene har mulighet til å ha egne wiki-sider, og det er

[3] https://about.gitlab.com/product/continuous-integration/
[4] https://about.gitlab.com/product/continuous-integration/

mulig å koble opp prosjektet mot Trello, for å kunne planlegge med teamet. Bitbucket lar deg også forgrene og slå sammen koden. Bitbucket bruker også et lagringssystem som heter Git Large File Storage, som gir raskere fetch- og kloningstid for større filer.
Bitbucket er gratis, men har to betalingsmetoder for ekstra funksjonalitet:

- 2$/mnd/bruker gir deg muligheten til å ha mer enn 5 brukere på et prosjekt.
- 5$/mnd/bruker gir deg:
    - Merge checking, som gir deg muligheten til å sette noen krav til code merge, og lar deg samkjøre mergen med code reviews.
    - IP-whitelisting, som lar deg velge hvem IP-Adresse som har tilgang til prosjektet.
    - Two-step verification, som gjør prosjektet tryggere ved å kreve et ekstra steg i påloggingen.
    - Smart mirroring, gjør prosjektkloningstider raskere for store prosjekter.

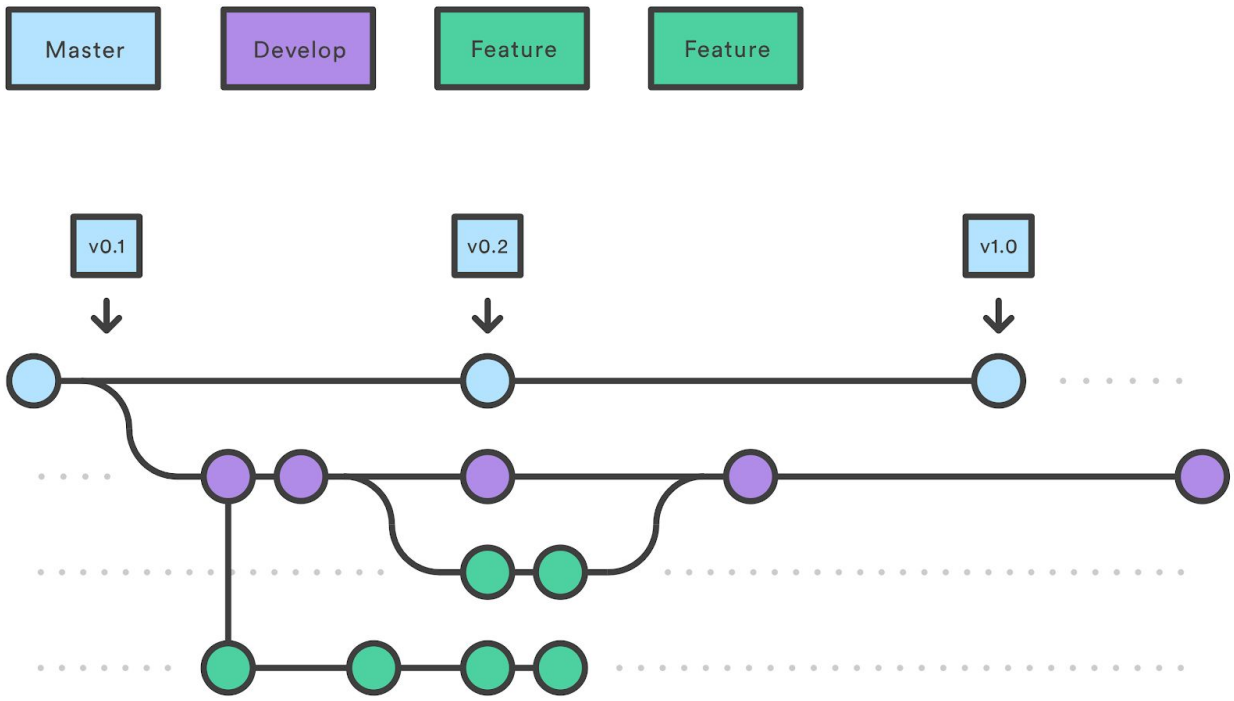
Kilder:

- https://en.wikipedia.org/wiki/Bitbucket
- https://bitbucket.org/product/pricing
- https://confluence.atlassian.com/bitbucket/git-large-file-storage-in-bitbucket-829078514.html
- https://confluence.atlassian.com/bitbucket/suggest-or-require-checks-before-a-merge-856691474.html
- https://confluence.atlassian.com/bitbucket/control-access-to-your-private-content-862621261.html
- https://confluence.atlassian.com/bitbucketserver/smart-mirroring-776640046.html

3.5. Source Forge

Source Forge[1] er et gratis system for å styre et samarbeid om programvareutvikling. Det er en webbasert tjeneste for å hjelpe til med å utvikle og distribuere programvare. Det er en forutsetning at programvaren er gratis for alle og tilbys som åpen kildekode (Free and open source software (FOSS))[2].

Websiden tilbyr et versjonkontrollsystem[3] med bugrapporteringssystem. Programvaren som utvikles kan lastes ned fra websiden og det oppfordres til å skrive en wiki til dokumentasjon og oppdatere en mikroblogg når det kommer oppdateringer[1]. Utviklere kan velge mellom kontrollsystemer som Git, Mercurial og CVS[4].



Figur 3.5.1: Typisk arbeidsflyt i programvareutvikling[7]

Git brukes ofte i programvareutviking og da også mye hos Source Forge. Figur 3.5.1 viser hvordan utviklere “brancher” ut til forskjellige forgreininger. Senere i utviklingen “merger” man forgreiningene sammen igjen.

Programvaren kan enten tilbys til nedlasting for installering eller den kan publiseres som webtjeneste under subdomene under sourceforge.net med grati bruk og tilgang til MySQL database[4].

I en perioden mellom 2013 og 2016[1] har Source Forge bundlet annen programvare sammen med det som ble lastet ned også uten utvikleren sin tillatelse[6]. Det var ikke veldig populært og gikk hardt utover ryktet.

SourceForge får inntekter av annonser på siden. Med 33 millioner brukere og 4.5 millioner nedlastinger hver dag[4] er det potensiale for store inntekter. Andre alternativer er GForge og FusionForge[5].

Kilder:

[1] https://en.wikipedia.org/wiki/SourceForge
[2] https://en.wikipedia.org/wiki/Free_and_open-source_software
[3] https://en.wikipedia.org/wiki/Repository_(version_control)
[4] https://sourceforge.net/
[5] https://en.wikipedia.org/wiki/Forge_(software)

[6]

https://www.pcworld.com/article/3032490/new-sourceforge-owners-kill-contentious-devshare-bloatware-program.html

[7] https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

3.5. BitKeeper

BitKeeper var den originale distribuerte kildekode-systemet, som Linus Torvald, skaperen av Linux Kernel (2002-2005) brukte